On efficiently characterizing solutions of linear diophantine equations and its application to data dependence analysis

Christine Eisenbeis, Olivier Temam, Harry Wijshoff

UNITÉ DE RECHERCHE
INRIA-RENNES

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P.105
78153 Le Chesnay Cedex
France
Tél.:(1) 39 63 55 11

Rapports de Recherche

1992 25ème anniversaire

N° 1616

Programme 1
Architectures parallèles, Bases de données,
Réseaux et Systèmes distribués

ON EFFICIENTLY CHARACTERIZING SOLUTIONS OF LINEAR DIOPHANTINE EQUATIONS AND ITS APPLICATION TO DATA DEPENDENCE ANALYSIS

Christine EISENBEIS
Olivier TEMAM
Harry WIJSHOFF

Février 1992

RR-1616

IRISA

INSTITUT DE RECHERCHE EN INFORMATIQUE
ET SYSTEMES ALEATOIRES

Campus Universitaire de Beaulieu
35042 - RENNES CEDEX FRANCE
Tél. : 99 84 71 00 - Télex : UNIRISA 950 473 F
Télécopie : 99 38 38 32

On efficiently characterizing solutions of linear Diophantine equations and its application to data dependence analysis*

Christine Eisenbeis† Olivier Temam‡ Harry Wijshoff§

January 28, 1992

Publication Interne n° 633 - Janvier 1992 - 22 pages - Programme 1

Abstract

In this paper we present several sets of mathematical tools for characterizing the solutions of linear Diophantine equations. First, a number of methods are given for reducing the complexity of the computations. Thereafter, we introduce different techniques for determining the exact number of solutions of linear Diophantine equations. Finally, we present a method for extracting efficiently the solutions of such equations. For all these methods, the main focus has been put on their applicability and efficiency for data dependence analysis.

Une caractérisation efficace des solutions des équations Diophantiennes linéaires et application à l'analyse de dépendances de données.

Résumé

Dans ce papier, nous présentons un ensemble d'outils mathématiques permettant de caractériser les solutions d'équations Diophantiennes linéaires. Dans un premier temps, nous montrons comment réduire la complexité des problèmes posés. Puis, nous introduisons plusieurs techniques pour calculer le nombre exact de solutions d'une équation Diophantienne linéaire. Enfin, nous montrons comment extraire, de manière efficace, les solutions de ces équations. Toutes ces méthodes ont été élaborées de telle façon qu'elles puissent être appliquées efficacement aux problèmes de dépendance de données.

Keywords: linear Diophantine equation, data dependence, data locality, dependence test, number theory

*Part of this work was done while the two authors O. Temam and H. Wijshoff were employed by the Center for Supercomputing Research and Development, University of Illinois at Urbana-Champaign, USA.

†INRIA, Domaine de Voluceau, 78153 Le Chesnay CEDEX, France

‡IRISA/INRIA, Campus de Beaulieu, 35042 Rennes CEDEX, France

§Department of Computer Science, Utrecht University, Padualaan 14, 3584 CH Utrecht, the Netherlands

1 Introduction

The extensive use of parallelism, fast processors and hierarchical memory systems greatly enhance the performance potential for modern architectures. However, compiler designers and programmers face the difficult task of making optimal use of these architectural improvements. One of the most crucial bottlenecks for the performance of these architectures consists of efficient memory management. The first step to realize efficient memory management is a good understanding of the memory behavior of programs. A major part of the codes which run on these high-performance systems spend a large fraction of their execution time in loops. Therefore, it is reasonable to restrict our study to the memory behavior, i.e., memory references, exhibited by these structures. These references consist mainly of references to arrays. The analysis of array references is equivalent to that of array subscript functions. Studies such as in [ShLY89] show that most subscripts are linear functions of loop indexes. Therefore, most problems related to reference analysis can be formulated as the characterization of the solutions of one or a set of linear Diophantine equations.

Let us consider the loop nest of figure 1. In order to study the data dependencies within this loop nest

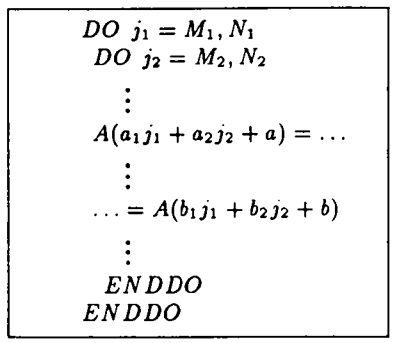

Figure 1: Fortran DO loop nest

equation $a_1 j_1 + a_2 j_2 + a = b_1 j_1' + b_2 j_2' + b$ has to be solved. Similarly, understanding the data locality of array A requires the analysis of equations $a_1 j_1 + a_2 j_2 + a = I$ and $b_1 j_1 + b_2 j_2 + b = I$ (where I is a location of array A). In both cases, we can reduce the analyses of these subscripts to the characterization of the solutions of the single linear Diophantine equation:

$$a_1 j_1 + \ldots + a_n j_n = C \tag{1}$$

where n is the number of variables and C is a constant.

Data dependence analysis has been studied to a great detail in the past. However, most of this research concentrated on fast techniques for determining whether there exists a data dependence or not, i.e., whether there is a solution for the Diophantine equation or not. With the necessity of explicitly managing data for utilizing cache, local memories, or explicit data prefetching, the need arises for an explicit characterization of the solution set and its size. So, in this paper we will concentrate on the "quantitative" analysis of these equations.

Depending on the problem considered, different characterizations are required: either the number of solutions has to be determined or the solutions themselves have to be enumerated. In the literature on number theory and other domains of mathematics, many studies can be found on the analysis of linear Diophantine equations. Because in this paper we study these equations in the context of data dependence analysis, the constraints imposed are slightly different from those considered in standard linear Diophantine equations analysis. First of all, the complexity of the computations required to obtain the number of solutions or the solutions themselves must be as small as possible. This enables the utilization of these techniques at compile-time and even at run-time. The latter is crucial since many of the coefficients of the Diophantine equations will not be known at compile-time. Secondly, we do not need to solve equations for any dimension n, but for small values of n only. This is caused by the fact that n represents the depth of a loop nest, which is small in general.

In this paper, fast methods for computing the exact or approximate number of solutions, and for enumerating the solutions are presented. Section 2 describes efficient techniques to compute the number of solutions, and is subdivided into three subsections which successively describe the formalization and

simplification of the problem, the methods for obtaining the exact number of solutions, and finally, approximation techniques for this number. In section 3 the enumeration of the solutions is studied. This section is subdivided into two subsections which successively describe the geometrical shape of the solution set, and a way to generate the solutions. In section 4 we consider the different possible applications of the results presented in the previous sections. Finally, in section 5, experimental results are presented.

2 Computing the number of solutions

Among the two problems considered, i.e., computing quickly and efficiently the number of solutions and generating the solutions, the first one is by far the most complex. Therefore, it is vital to simplify the problem as much as possible in order to obtain efficient computations. The modifications proposed constitute a set of tools aimed at simplifying the analysis of Diophantine equations. Some of them have other applications than only speeding up the computations presented further on.

Let us now give the exact formulation of the problem considered:

Problem 2.1

$$(M_1, N_1), \ldots, (M_n, N_n) \in \mathbf{Z} \times \mathbf{Z}$$
$$j_i \in [M_i, N_i]$$
$$a_1, \ldots, a_n, C \in \mathbf{Z}$$

$$a_1 j_1 + \ldots + a_n j_n = C \tag{2}$$

Find the number of integer solutions of equation (2) under the above constraints.

Remarks

- Throughout the next sections, we will refer to one specific example in order to illustrate the transformations and the results obtained. The example is the following:
  Example

n	M_1	N_1	a_1	M_2	N_2	a_2	M_3	N_3	a_3	C
3	-130	-36	30	15	-18	-36	50	240	40	1452

2.1 Problem reductions

In this section, three main transformations of the initial problem are presented. The equation is modified so that all coefficients are positive, and all lower bounds of the variables are equal to 0. Secondly, it is shown how a system of equations can be transformed into one single equation. Then, we try to reduce the value of the different coefficients, since these values determine the efficiency of the resolution methods. Finally, it is shown that the number of solutions of the problem with no bounds on the variables is related to the same problem with bounded variables.

2.1.1 Collapsing a set of equations

In many cases, the memory references which appear within a loop nest are due to multi-dimensional arrays. Let us consider the example of figure 2. In this case, finding the data dependencies for instance, would require solving equations

$$b_1 j_1 + b_2 j_2 + b = d_1 j_1' + d_2 j_2' + d$$

and

$$c_1 j_1 + c_2 j_2 + c = e_1 j_1' + e_2 j_2' + e$$

That is, a set of equations instead of the preferred single equation. In Fortran, multi-dimensional arrays are stored contiguously in column major order. For instance, element (i, j) of matrix A in our example, is stored in address *offset* + $(i + n_1 j)$ where n_1 is the first dimension of A. Therefore, it is possible to *linearize* all matrices into one-dimensional arrays and handle both of the equations at the same

```
DO j1 = M1, N1
 DO j2 = M2, N2
   ⋮
A(b1 j1 + b2 j2 + b, c1 j1 + c2 j2 + c) = ...
   ⋮
... = A(d1 j1 + d2 j2 + d, e1 j1 + e2 j2 + e)
   ⋮
 ENDDO
ENDDO
```

Figure 2: Loop nest computing on a 2-dimensional array

time. Consequently, instead of solving a set of equations, only one equation, derived from the linearized subscripts, has to be solved. In the previous example, equation

$$(b_1 j_1 + b_2 j_2 + b) + n_1(c_1 j_1 + c_2 j_2 + c) = (d_1 j_1' + d_2 j_2' + d) + n_1(e_1 j_1' + e_2 j_2' + e)$$

replaces the previous set of equations.

Remarks

- The number of unknowns in the *collapsed* equation is the same as in the original system of equations.
- It is commonly said that complex analysis of Diophantine equations is not necessary because coefficients found in matrix subscripts are generally simple [ShLY89], and the use of some simple heuristics should be sufficient. However, such transformations as the one proposed above cause the coefficients of the new Diophantine equation to have non-trivial values necessitating a more thorough analysis.

The linearization of a set of equations as described above does not only create large coefficients, but in many cases these coefficients will be composite numbers, causing the analysis of the equations to be very time consuming and complex.

A way to overcome this disadvantage is to use prime multipliers instead of the leading dimension of the arrays for linearizing a system of equations. This can be achieved by viewing the set of equations:

$$\begin{cases} f_1(\vec{i}) = 0 \\ f_2(\vec{i}) = 0 \\ \vdots \\ f_k(\vec{i}) = 0 \end{cases} \tag{3}$$

as:

$$A\vec{i} = \vec{c}$$

with A a $k \times n$ integer matrix. Assume, for simplicity's sake, that $n = k$ and A is non-singular. Then, this system can be transformed by applying row eliminations on A (Gaussian Elimination) to:

$$D\vec{i} = \vec{c'}$$

with D a $k \times k$ diagonal integer matrix. Write this system as:

$$\begin{cases} f_1'(\vec{i}) = 0 \\ f_2'(\vec{i}) = 0 \\ \vdots \\ f_k'(\vec{i}) = 0 \end{cases} \tag{4}$$

Note that every equation uses a different subscript of $\vec{i}$.

Define

$$M = max_{j,\, \vec{i} \in [M_1,N_1] \times \ldots \times [M_n,N_n]} \left| f_j'(\vec{i}) \right|$$

and the sequence $p_1, \ldots, p_k$ of prime numbers such that p_1 is the smallest prime greater than M, p_2 is the smallest prime greater than p_1, and so on. Then, solving the system of equations (4) is equivalent to solving:

$$p_1 f_1'(\vec{i}) + p_2 f_2'(\vec{i}) + \ldots + p_k f_k'(\vec{i}) = \vec{0} \quad (5)$$

Note that $det(A)$ directly bounds the growth of the coefficients in equation (5).

The assumption $k = n$ is mostly not satisfied, however. In general, the original system will be rectangular, with $k < n$. In this case, it is much harder to transform the orginal system $f_1(\vec{i}), \ldots, f_n(\vec{i})$ to $f_1'(\vec{i}), \ldots, f_n'(\vec{i})$ such that the index sets $\vec{i}$ of the f_i' are independent of each other. One way of obtaining this would be to allow "column" eliminations on A. However, contrary to the above mentioned, the resulting problem would consist of solving: $D(\vec{j}) = \vec{c}$ and $U(\vec{i}) = \vec{j}$, with D a $k \times k$ diagonal matrix and U a $n \times n$ upper triangular matrix.

This latter equation complicates the analysis of the first one, as there is no one to one correspondence between the $\vec{j}$ and $\vec{i}$ vectors in general. In the special case of $n = k + 1$ this one to one correspondence exists and this method can be applied. In fact for the above described transformations to obtain a diagonal submatrix, the "Smith normal form" can also be used [N72].

2.1.2 -A few simple modifications

The simplifications proposed below are trivial ones which only aim at simplifying the constraints of Problem 2.1.

- *Make all* $a_i > 0$. If $a_i < 0$, write $a_i j_i = |a_i| j_i'$ with $j_i' = -j_i$, i.e., $j_i' \in [-N_i, -M_i]$. Therefore, through a modification on the bounding intervals of the j_i, it is now possible to assume that $a_i > 0, \forall i \in \{1, \ldots, n\}$.

  Example
  The example of section 2 becomes:

n	M_1	N_1	a_1	M_2	N_2	a_2	M_3	N_3	a_3	C
3	-130	-36	30	-18	15	36	50	240	40	1452

- *Make all* $M_i = 0$. If $M_i \neq 0$, write $j_i' = j_i - M_i$, i.e., $j_i' \in \left[0, N_i'\right]$, with $N_i' = N_i - M_i$. Equation (2) now becomes

  $$a_1 j_1 + \ldots + a_i j_i' + \ldots + a_n j_n = C'$$

  with $C' = C - a_i M_i$. Therefore, through a modification on C, we can now assume that $M_i = 0, \forall i \in \{1, \ldots, n\}$.

  Example
  The example of section 2 becomes:

n	M_1	N_1	a_1	M_2	N_2	a_2	M_3	N_3	a_3	C
3	0	94	30	0	33	36	0	190	40	4000

- *Divide equation (2) by the gcd of the coefficients.* Let $d_{1\ldots n}$ denote the gcd of $a_1, \ldots, a_n$ $(\bigwedge_{i=1}^{n} a_i)$. If $d_{1\ldots n}$ does not divide C then there is no solution to equation (2). Otherwise, write $C' = \frac{C}{d_{1\ldots n}}$ and $a_i' = \frac{a_i}{d_{1\ldots n}}$, $\forall i 1 \leq i \leq n$. Therefore, through a modification on C, it is now possible to assume that $\bigwedge_{i=1}^{n} a_i = 1$.
  It must be noted that the gcd of several integers can be computed very quickly using Euclid's algorithm [GKP89].

  Example
  In this case, $gcd(a_1, a_2, a_3) = 2$. So, the example of section 2 becomes:

n	M_1	N_1	a_1	M_2	N_2	a_2	M_3	N_3	a_3	C
3	0	94	15	0	33	18	0	190	20	2000

2.1.3 Unbounding the variables

A more complex modification of the problem is to unbound variables j_i, that is, to have $j_i \geq 0$ instead of $j_i \in [0, N_i]$. The reason for unbounding these variables is twofold. First, the bounds on the variables make the problem much more difficult to deal with. Essentially, because of the greater number of unknowns and the fact that the solution space is finite, the complexity of finding solutions increases. Second, most number theory results on linear Diophantine equations do not take into account such bounds, and therefore cannot be applied to the problem considered.

Let $\Delta_{N_1,\ldots,N_n}(a_1,\ldots,a_n;C)$ be the number of solutions of Problem 2.1, and $\Delta_\infty(a_1,\ldots,a_n;C)$ be the number of solutions of the same problem without bounds on variables j_i. We will show that $\Delta_{N_1,\ldots,N_n}(a_1,\ldots,a_n;C)$ can be expressed as a function of $\Delta_\infty(a_1,\ldots,a_n;C)$.

Let us first recall that any rational fraction $\frac{P(x)}{Q(x)}$ where $P(x), Q(x)$ are polynomials, can be developed into an infinite series: $\sum_{i=0}^{\infty} c_i x^i$. Then, a well known result in number theory is that $\Delta_\infty(a_1,\ldots,a_n;C)$ is the coefficient of x^C in the series development of $\frac{1}{(1-x^{a_1})\ldots(1-x^{a_n})}$, and that $\Delta_{N_1,\ldots,N_n}(a_1,\ldots,a_n;C)$ is the coefficient of x^C in the series development of $\frac{(1-x^{(N_1+1)a_1})\ldots(1-x^{(N_n+1)a_n})}{(1-x^{a_1})\ldots(1-x^{a_n})}$ [PS64]. Consider case $n = 2$, then

$$\frac{(1-x^{(N_1+1)a_1})(1-x^{(N_2+1)a_2})}{(1-x^{a_1})(1-x^{a_2})}$$
$$= \frac{1}{(1-x^{a_1})(1-x^{a_2})} - \frac{x^{(N_1+1)a_1}}{(1-x^{a_1})(1-x^{a_2})} - \frac{x^{(N_2+1)a_2}}{(1-x^{a_1})(1-x^{a_2})} + \frac{x^{(N_1+1)a_1+(N_2+1)a_2}}{(1-x^{a_1})(1-x^{a_2})}$$

Therefore,

$$\begin{aligned}&\Delta_{N_1,N_2}(a_1,a_2;C)\\ &= \Delta_\infty(a_1,a_2;C) - \Delta_\infty(a_1,a_2;C-(N_1+1)a_1)\\ &-\Delta_\infty(a_1,a_2;C-(N_2+1)a_2) + \Delta_\infty(a_1,a_2;C-(N_1+1)a_1-(N_2+1)a_2)\end{aligned}$$

Similarly, $(1-x^{(N_1+1)a_1})\ldots(1-x^{(N_n+1)a_n})$ can be written, for any n, as $\sum_{i=0}^{2^n}\gamma_i x^{\delta_i}$, that is, a polynomial of 2^n terms. Therefore, $\Delta_{N_1,\ldots,N_n}(a_1,\ldots,a_n;C)$ can be expressed as $\sum_{i=0}^{2^n}\gamma_i\Delta_\infty(a_1,\ldots,a_n;C-\delta_i)$, an expression with 2^n different terms of $\Delta_\infty(a_1,\ldots,a_n;C)$.

Since the values of n considered ($1 \leq n \leq 4$) are small enough, 2^n ($2 \leq 2^n \leq 16$) will not be too large, rendering the computations of $\Delta_\infty(a_1,\ldots,a_n;C)$ feasible. So, it is possible to restrict our effort to determine the expression of $\Delta_\infty(a_1,\ldots,a_n;C)$.

The new problem can be expressed as follows:

Problem 2.2

$$j_i \geq 0$$
$$a_1,\ldots,a_n,C \in \mathbf{N}$$

$$a_1 j_1 + \ldots + a_n j_n = C \tag{6}$$

Find the number of integer solutions of equation (6) under the above constraints, that is, compute $\Delta_\infty(a_1,\ldots,a_n;C)$.

2.1.4 Reducing the value of the coefficients

Reducing the value of the coefficients a_i and C is important since the complexity of the solution techniques is related to the size of these parameters. Erhart [E64] proposed a method for reducing these values by dividing the coefficients by appropriate values. The constraints of Erhart Problem are similar to those of Problem 1.2 except that all the variables (let us call them j_i') must be strictly positive:

Erhart Problem

$$j_i' > 0$$
$$a_1,\ldots,a_n,C' \in \mathbf{N}$$

$$a_1 j_1' + \ldots + a_n j_n' = C' \tag{7}$$

Find the number of integer solutions of the previous equation under the above constraints.

This small difference in the constraints does not prevent us from applying Erhart results to Problem 1.2, similarly to the techniques of section 2.1.2, let us define $j_i' = j_i + 1$ with $j_i \geq 0$. Then $j_i' > 0$, and problem 1.2 can be reformulated as follows:

Problem 2.3

$$j_i' > 0$$
$$a_1, \ldots, a_n, C \in \mathbf{N}$$

$$a_1 j_1' + \ldots + a_n j_n' = C + (a_1 + \ldots + a_n) = C'$$

Find the number of integer solutions of the previous equation under the above constraints.

Let us now give Erhart theorem:

Theorem 2.1 *Let $a_1, \ldots, a_n$ be relatively prime. Let us define $g_i = \bigwedge_{k=1, k\neq i}^{n} a_k$, and $A_i = \frac{a_i}{\prod_{k=1, k\neq i}^{n} g_k}$. Let C_n' be defined as follows ($k_i \geq 0$):*

- $C_1' g_1 = C' + a_1 k_1$ *where $C_1' g_1$ is the smallest multiple of g_1 of this form,*
- $C_2' g_2 = C_1' + \frac{a_2}{g_1} k_2$ *where $C_2' g_2$ is the smallest multiple of g_2 of this form,*
- $C_3' g_3 = C_2' + \frac{a_3}{g_1 g_2} k_3$ *where $C_3' g_3$ is the smallest multiple of g_3 of this form,*

$\vdots$

- $C_n' g_n = C_{n-1}' + \frac{a_n}{g_1 \cdots g_{n-1}} k_n$ *where $C_n' g_n$ is the smallest multiple of g_i of this form.*

Then we can reduce problem 2.3 to:

Problem 2.4

$$j_i' > 0$$
$$A_1, \ldots, A_n, C_n' \text{ as defined above}$$

$$A_1 j_1' + \ldots + A_n j_n' = C_n' \tag{8}$$

Find the number of integer solutions of equation (8) under the above constraints.

Remarks
- It must be noted that this algorithm has a small finite number of steps equal to n.
- The problem of finding C_i' and k_i on each iteration is equivalent to finding the Bezout [GKP89] numbers of equation $C_i' u_1 - \frac{a_i}{g_1 \cdots g_{i-1}} u_2 = 1$. Then $g_i = u_1$ and $k_i = u_2$.

Example
When applying Erhart reduction to the example of section 2, we obtain:

- $g_1 = 2$, $A_1 = 1$, $k_1 = 0$, $C_1' = 1052$.
- $g_2 = 5$, $A_1 = 3$, $k_2 = 2$, $C_2' = 214$.
- $g_3 = 3$, $A_1 = 2$, $k_3 = 1$, $C_3' = 72$.

n	M_1	N_1	a_1	M_2	N_2	a_2	M_3	N_3	a_3	C
3	1		1	1		3	1		2	72

This reduction method ensures smaller values of the coefficients and the right-hand side of the equation. It must be noted that the efficiency of this technique is highly dependent on the values of the g_i.

2.2 Fast methods for determining the exact number of solutions

Let us now consider Problem 2.2, with all previous simplifications already performed (Erhart simplifications are not compulsory since they are independent of the following computations). Three different methods are presented in this section. The first method is a new technique while the last two methods are based on work by Erhart and Comtet.

2.2.1 Fast determination of the number of solutions

The technique as presented here is based on a simple calculation. For our purposes, this technique is probably the most interesting one among those proposed here, because it is general enough to be applied for any value of n, and it is very efficient for $1 \leq n \leq 4$.

In the next paragraphs, the computation of $\Delta_\infty(a_1, \ldots, a_n; C)$ is described for $n \in \{2, 3, 4\}$. The method can be directly generalized for greater values of n.

Case $n = 2$ Problem 2.2 can be written as follows:

Problem 2.5

$$j_1 \geq 0, j_2 \geq 0 \tag{9}$$

$$a_1, a_2, C \in \mathbf{N} \tag{10}$$

$$a_1 \wedge a_2 = 1 \tag{11}$$

$$a_1 j_1 + a_2 j_2 = C \tag{12}$$

Find the number of integer solutions of equation (12) under the above constraints.

Hypothesis (11) is due to the simplification of section 2.1.2. Under these constraints, it is possible to apply Bezout's theorem to equation (12). This theorem states that there exist two integers u_1, u_2 such that

$$a_1 u_1 + a_2 u_2 = 1 \tag{13}$$

and that all solutions of equation (12) have the following expression:

$$\begin{aligned} 0 \leq \quad j_1 \quad &= C u_1 - a_2 \lambda \\ 0 \leq \quad j_2 \quad &= C u_2 + a_1 \lambda \end{aligned}$$

where λ is an integer parameter. The two constraints $j_1 \geq 0, j_2 \geq 0$ can be transformed into two constraints on λ (recall that $a_1 > 0$ and $a_2 > 0$):

$$\begin{aligned} \lambda \quad &\leq \quad \frac{C u_1}{a_2} \\ \lambda \quad &\geq \quad \frac{-C u_2}{a_1} \end{aligned}$$

Since λ is an integer, the number of solutions of equation (12), i.e., the number of possible values of λ, is given by the following expression:

$$\Delta_\infty(a_1, a_2; C) = \lfloor \frac{C u_1}{a_2} \rfloor - \lceil \frac{-C u_2}{a_1} \rceil + 1$$

Remarks

- u_1, u_2 can be computed very quickly using Euclid's algorithm. Above, it was mentioned that it is necessary to compute the gcd of a_1, a_2 in order to perform the simplifications. We indicated that Euclid's algorithm can be used for that purpose also. In fact, it is possible to compute u_1, u_2 *and* $d_{12} = gcd(a_1, a_2)$ at the same time, during one single execution of Euclid's algorithm [GKP89], which makes the computation of these three numbers very efficient.

Case $n = 3$ For $n = 3$, equation (2) can be written as follows:

$$a_1 j_1 + a_2 j_2 = C - a_3 j_3$$

Consequently,

$$\Delta_\infty(a_1, a_2, a_3; C) = \sum_{\text{all possible } j_3} \Delta_\infty(a_1, a_2; C - a_3 j_3)$$

Let us give the closed formula for the possible values of j_3:

if j_3 is a solution then $d_{12} \mid (C - a_3 j_3)$, i.e., $C - a_3 j_3 = 0 \bmod d_{12}$

$\Leftrightarrow\ C - a_3 j_3 \equiv 0\,(d_{12})$

$\Leftrightarrow\ j_3 \equiv a_3^{-1} C\,(d_{12})$, since $a_3 \wedge d_{12} = 1$

Define $j_3^0 = a_3^{-1} C \bmod d_{12}$, then the possible values of j_3 are of the form $j_3 = j_3^0 + \lambda d_{12}$ (where λ is an integer parameter). There are two constraints on j_3 ($j_3 \geq 0$, $C - a_3 j_3 \geq 0$) which give the interval within which λ varies, i.e., $\lambda \in \left[0, \lfloor \frac{C - a_3 j_3^0}{a_3 d_{12}} \rfloor\right]$. Therefore,

$$\Delta_\infty(a_1, a_2, a_3; C) = \sum_{\lambda=0}^{\lfloor \frac{C - a_3 j_3^0}{a_3 d_{12}} \rfloor} \Delta_\infty(a_1, a_2; (C - a_3 j_3^0) - \lambda a_3 d_{12})$$

Example

The example of section 2 can be treated as follows:

- $j_3^0 = 1$.
- $\lambda \in [0, 33]$.
- The number of integer solutions of the unbounded equation is 397. For the bounded equation, the number of solutions is 168.

Case $n = 4$ For $n = 4$, equation (2) can be written as follows:

$$a_1 j_1 + a_2 j_2 = C - a_3 j_3 - a_4 j_4$$

For any value of (j_1, j_2) and (j_3, j_4), there are two integer parameters μ and ν such that

$$a_1 j_1 + a_2 j_2 = \mu d_{12} \tag{14}$$

$$a_3 j_3 + a_4 j_4 = \nu d_{34} \tag{15}$$

For given values of μ and ν, there are respectively $\Delta_\infty(a_1, a_2; \mu d_{12})$ pairs (j_1, j_2) solutions of (14), and $\Delta_\infty(a_3, a_4; \nu d_{34})$ pairs (j_3, j_4) solutions of (15). Since $\mu d_{12} = C - \nu d_{34}$, for each value of μ there exists at most one single value for ν. Now, for each of pair (j_1, j_2), solution of $a_1 j_1 + a_2 j_2 = \mu d_{12}$ and each pair (j_3, j_4), solution of $a_3 j_3 + a_4 j_4 = C - \mu d_{12}$, the 4-tuplet (j_1, j_2, j_3, j_4) constitutes a solution to equation (12). Therefore, for a given value of μ, there are $\Delta_\infty(a_1, a_2; \mu d_{12}) \times \Delta_\infty(a_3, a_4; C - \mu d_{12})$ solutions to equation (2). Let us now give the expression of all possible values of μ:

if μ is a solution then $d_{34} \mid (C - \mu d_{12})$, since $C - \mu d_{12} = \nu d_{34}$

$\Leftrightarrow\ C - \mu d_{12} \equiv 0\,(d_{34})$

$\Leftrightarrow\ \mu \equiv d_{12}^{-1} C\,(d_{34})$, since $d_{34} \wedge d_{12} = 1$

Let $\mu^0 = d_{12}^{-1} C \bmod d_{34}$, then the possible values of μ are of the form $\mu = \mu^0 + \lambda d_{34}$. There are two constraints on μ ($\mu \geq 0$, $C - d_{12}\mu \geq 0$), which give the interval within which λ varies. That is, $\lambda \in \left[0, \lfloor \frac{C - d_{12}\mu^0}{d_{12} d_{34}} \rfloor\right]$.

Therefore,

$$\Delta_\infty(a_1, a_2, a_3, a_4; C) = \sum_{\lambda=0}^{\lfloor \frac{C - d_{12}\mu^0}{d_{12} d_{34}} \rfloor} \Delta_\infty(a_1, a_2; C - \mu^0 d_{12} - \lambda d_{12} d_{34}) \times \Delta_\infty(a_1, a_2; \mu^0 d_{12} + \lambda d_{12} d_{34})$$

Remarks

- The method described above can be generalized for any value of n, except that instead of a formula or a single sum, several nested sums will have to be computed to obtain $\Delta_\infty(a_1, \ldots, a_n; C)$. So, the complexity of the computations will grow exponentially with n because of the nesting of the sums.
- However, the level of nesting of the sums is equal to $\frac{n-2}{2}$, which suggests that the method is still usable for $n = 5, 6$ at a relatively moderate cost.
- The integer inverse of numbers such as ${a_3}^{-1}\ (d_{12})$ can be computed very quickly using Euclid's algorithm. For instance, ${a_3}^{-1}\ (d_{12})$ is an integer such that ${a_3}^{-1}a_3 \equiv 1\ (d_{12})$, i.e., such that $\exists u \in \mathbf{N} : {a_3}^{-1}a_3 = 1 + ud_{12}$, which can be rewritten as ${a_3}^{-1}a_3 - ud_{12} = 1$. Therefore, finding ${a_3}^{-1}\ (d_{12})$ is strictly equivalent to computing the Bezout numbers of equation ${a_3}^{-1}j_1 - d_{12}j_2 = 1$, which justifies the use of Euclid's algorithm.

Complexity of the method The simplifications presented in section 2.1 and the use of Euclid's algorithm have, in general, a menial cost compared to that of the above method, and will therefore be neglected. The complexity of the computation depends mainly on $\lfloor \frac{C - a_3 j_3^0}{a_3 d_{12}} \rfloor$ for $n = 3$, and on $\lfloor \frac{C - d_{12}\mu^0}{d_{12}d_{34}} \rfloor$ for $n = 4$. However, it can be shown that the complexity can be changed such that it depends on the coefficients only, and not on C, which can be very large in some cases (cf. A.1 for the expressions of Δ_∞).

For $n = 2$, the number of operations is nearly negligible (less than 10). For n equals 3, or 4, Δ_∞ is a sum of elementary operations and therefore, the complexity of the computations is equal to the number of elementary operations for each iteration of the sum, times the boundaries of the sum, times 2^n, since 2^n values of Δ_∞ must be computed in order to get the number of solutions of Problem 1.

Thus, for $n = 3$, the complexity is of the form

$$2^3 \times \alpha_3 \times \lfloor \frac{C - a_3 j_3^0}{a_3 d_{12}} \rfloor \simeq 2^3 \times \alpha_3 \times \lfloor \frac{C}{a_3 d_{12}} \rfloor \tag{16}$$

while for $n = 4$, the complexity is

$$2^4 \times \alpha_4 \times \lfloor \frac{C - d_{12}\mu^0}{d_{12}d_{34}} \rfloor \simeq 2^4 \times \alpha_4 \times \lfloor \frac{C}{d_{12}d_{34}} \rfloor \tag{17}$$

with α_3 and α_4 less or equal to 12.

For the transformed expressions of Δ_∞ which can be found in A.1, the complexity, for $n = 3$, is given by

$$2^3 \times \beta_3 \times (a_1 + a_2) \tag{18}$$

while for $n = 4$, it is given by

$$2^4 \times \beta_4 \times (a_1a_3 + a_1a_4 + a_2a_3 + a_2a_4) \tag{19}$$

with β_3 and β_4 less or equal to 20.

Since there are no constraints on the coefficients a_i, they can be reordered so that a_3 is the largest one reducing the complexity for the case $n = 3$ (16). Erhart reduction gives us means for reducing C, but it also reduces the value of the coefficients, therefore its use for reducing the complexity of (16) and (17) should be handled with care.

For the modified expressions of the sums, the complexity of (18) and (19) depend on the coefficients a_i only. Similarly, the coefficients should be reordered so that $a_1 + a_2$ and $a_1a_3 + a_1a_4 + a_2a_3 + a_2a_4$ are as small as possible. On the other hand, it can be noticed that, this time, Erhart reduction can be used straightforwardly because parameter C does not appear in the complexity bound.

Finally, it must be noted that the fact that the exact number of operations of both methods can be computed at a very low cost suggesting the use of an heuristic for determining which type of sums should be used in each case.

2.2.2 Two other ways for determining the exact number of solutions

Erhart [E64] and Comtet [C80] proposed two different methods for determining the exact number of solutions. These methods are interesting but have some constraints which prevent their systematic use. Both methods are based on the observation that $\Delta_\infty(a_1, \ldots, a_n; C)$ is a quasi-polynomial[1].

[1] a quasi-polynomial is a function of the form $f(x) = \sum_{i=0}^{N} c_i(x)x^i$ where $c_i(x)$ is a periodic function with an integer period

Erhart's method When the a_i are pairwise prime, Erhart proved that $\Delta_\infty(a_1,\ldots,a_n;C)$ can be decomposed as follows:

$$\Delta_\infty(a_1,\ldots,a_n;C) = f(C) + \psi(C)$$

where $f(C)$ is a polynomial of degree $n-1$ and $\psi(C)$ is a periodic function of period $a_1 \times \ldots \times a_n$:

$$\psi(C) = \sum_{i=1}^{n} \phi_{a_i}(C)$$

with $\phi_{a_i}(C)$ a periodic function of period a_i:

$$\phi_{a_i}(C) = \begin{cases} \mathbf{a_i\ odd}: \\ \dfrac{1}{2^{k-2}a_i} \displaystyle\sum_{p=1}^{\frac{a_i-1}{2}} \dfrac{\cos\left[\frac{n-1}{2}\pi - \frac{2C+\sum_{j\neq i} a_j}{a_i} p\pi\right]}{\prod_{j\neq i} \sin\frac{\pi a_j}{a_i} p} \\ \mathbf{a_i\ even}: \\ \dfrac{(-1)^n}{2^{k-1}a_i} + \dfrac{1}{2^{k-2}a_i} \displaystyle\sum_{p=1}^{\frac{a_i-2}{2}} \dfrac{\cos\left[\frac{n-1}{2}\pi - \frac{2C+\sum_{j\neq i} a_j}{a_i} p\pi\right]}{\prod_{j\neq i} \sin\frac{\pi a_j}{a_i} p} \end{cases}$$

The problem with computing $\psi(C)$ is caused by the presence of costly functions such as *sinus, cosinus* in the expression of the function. The cost of computing this function is of the order of $\sum_{i=1}^{n} na_i$. On the other hand, for $n \leq 6$, Erhart determined explicitly the formulas for $f(C)$ (cf. A.2), which are simple.

The constraints on the a_i (a_i pairwise prime) are relatively strong. However, it is interesting to note that, for $n = 3$, the Erhart reduction of section 2.1.2 makes all coefficients pairwise prime. This is not true for greater values of n, though the reduction method still increases the probability of having pairwise prime coefficients.

Comtet method Comtet noticed that the quasi-period of $\Delta_\infty(a_1,\ldots,a_n;C)$ is equal to A, the least common multiple of the coefficients.

Using this fact, he proved that, for all $c, 0 \leq c < A$, for all C : $C \equiv c\,(A)$, $\Delta_\infty(a_1,\ldots,a_n;C)$ is a polynomial in C of degree $n-1$. Therefore, only the first n values of C with $C \equiv c\,(A)$, $C = c, c+A, \ldots, c+(n-1)A$, are needed to compute this polynomial. Consequently, $A \times n$ values of $\Delta_\infty(a_1,\ldots,a_n;C)$ need to be computed to get a complete characterization of this function, that is, to get the A polynomials corresponding to the A possible values of c.

For small values of C, the technique presented in section 2.2.1 is very efficient. Therefore combining this technique with the one proposed by Comtet yields a relatively efficient method. It must be noted that its performance is highly dependent on the value of A. Indeed, if $C \equiv c\,(A)$, then $\Delta_\infty(a_1,\ldots,a_n;C)$ must be computed for $C = c, c+A, \ldots, c+A(n-1)$, and if A is too large these values of C will be large, and therefore computing the initial values of $\Delta_\infty(a_1,\ldots,a_n;C)$ will be too costly. There again, Erhart reduction can be applied in order to reduce the value of C and $(a_i)_{1\leq i\leq n}$, and thereby the value of A.

2.2.3 Comparison of the different methods

The two concerns that prevail in the analysis of the above techniques are *efficiency* and *applicability*.

Comtet's method:

- This method must be used in combination with the fast determination technique in order to get the initial values of Δ_∞ needed to compute the coefficients of the polynomial.
- The technique works for small values of A and n only. Otherwise the number of terms of Δ_∞ that must be computed becomes prohibitive. Unfortunately, the value of A in the applications considered might often be large, especially if the *collapsed equation* of section 2.1.1 is used. A will also be large if the a_i are relatively prime (because A is then equal to the product of the a_i), but in that case Erhart's method can be used.

- If $n \times A$ is small enough, the technique is very interesting because it provides, at a moderate cost, the analytical expression of Δ_∞ as a collection of A polynomials of degree n.

Erhart's method:

- Erhart's method is probably the most elegant regarding mathematics, but the constraints on the coefficients are important, a fact which restricts its applicability. However, it is not too costly to test if the a_i are pairwise prime using Euclid's algorithm.
- The analytical expression of the function ψ is relatively complicated (presence of *cos* and *sin*), and therefore hardly usable.
- If $\sum_{i=1}^{n} a_i$ is small enough, the method can be used for $n \leq 6$, which is not a big constraint.
- The method is interesting in itself since the decomposition into a polynomial and a periodic function gives a good insight for the function Δ_∞.

Fast determination:

- This method is more straightforward compared to the two above, as it is based on a fast determination of the solutions.
- The analytical expressions obtained allow a good understanding of the role of the different parameters.
- For common values of the parameters and $n < 4$, the technique is at least as efficient as the two above. It can still be used for any other dimension, but with a loss of efficiency.
- This method imposes *no constraints at all* on the parameters.

2.3 Getting an approximation of the number of solutions

Computing an approximate number of solutions instead of the exact one enables the computation to be simplified considerably. In this section, approximations are derived of which the complexity is a function of n only, so, it is independent of the value of any other parameter.

The advantage of getting an approximate number of solutions is twofold. First, it must be noted that most methods provide us with *algorithms* for computing $\Delta_\infty(a_1, \ldots, a_n; C)$ while for approximations, analytical expressions can be derived. In the latter case, $\Delta_\infty(a_1, \ldots, a_n; C)$ can be studied as a function of C or the coefficients a_i. Second, depending on the application, it is not always necessary to compute an exact number of solutions. Sometimes, all we want to know is whether there are "lots" or "few" solutions.

We will present two different approximations, both of them are derived from the exact methods presented above. The error yielded by each approximation is highly dependent on the value of the parameters. Both approximations provide us with polynomial expressions.

2.3.1 Approximation derived from fast determination

The idea of this approximation is very simple. None of the computations are changed except that, in the final expressions $\lfloor X \rfloor$ is systematically replaced by $X - \frac{1}{2}$. Then, the sums can be computed analytically, and the result is a polynomial in C (cf. A.3 for more details):

Case $n = 3$:

$$\Delta_\infty(a_1, a_2, a_3; C) \simeq$$
$$\left[(C - a_3 j_3^0)(\tfrac{u_1}{a_2} + \tfrac{u_2}{a_1}) + 1\right](\tfrac{C - a_3 j_3^0}{a_3 d_{12}} + \tfrac{1}{2}) - d_{12} a_3 (\tfrac{u_1}{a_2} + \tfrac{u_2}{a_1}) \frac{(\frac{C - a_3 j_3^0}{a_3 d_{12}} - \frac{1}{2})(\frac{C - a_3 j_3^0}{a_3 d_{12}} + \frac{1}{2})}{2}$$

An upper bound on the error of this approximate expression of $\Delta_\infty(a_1, a_2, a_3; C)$ is $\lfloor \frac{C}{a_3 d_{12}} \rfloor$. Since 2^n values of Δ_∞ must be computed to get the number of solutions, an upper bound on the overall error is $2^n \times \lfloor \frac{C}{a_3 d_{12}} \rfloor$.

Example

The approximate number of solutions (for the bounded equation) of the example of section 2 is equal to 164.

2.3.2 Approximation derived from Erhart's method

As was pointed out, $\Delta_\infty(a_1,\ldots,a_n;C)$ can be considered as a sum of a polynomial and a periodic function. Experimental computations of the periodic function $\psi(C)$ tend to prove that its value is very small compared to that of the polynomial function $f(C)$, though it is quite hard to get a precise theoretical upper bound of $\psi(C)$. A gross upper bound of $\psi(C)$ is:

$$\sum_{i=1}^{n}\frac{1}{2^{k-1}a_i}+\frac{1}{2^{k-2}a_i}\times\frac{cos(\frac{n-1}{2}\pi-\frac{2C+\sum_{j\neq i}a_j}{a_i}(\frac{a_i-1}{4})\pi)\times\frac{cos(\frac{2C+\sum_{j\neq i}a_j}{a_i}(\frac{a_i+1}{2})\pi)}{cos(\frac{2C+\sum_{j\neq i}a_j}{2a_i})}-cos(\frac{n-1}{2}\pi)}{\prod_{j\neq i}min(sin\frac{\pi a_j}{a_i},sin\frac{\pi a_j}{a_i}\frac{a_i-2}{2})}$$

If this function is neglected, we get an approximate value of $\Delta_\infty(a_1,\ldots,a_n;C)$ using the polynomial function $f(C)$. Since this function $f(C)$ has been computed by Erhart for $1\leq n\leq 6$, it can be used for our purposes.

Example
The approximate number of solutions (for the bounded equation) of the example of section 2 is equal to 162.

2.3.3 Geometric characterization of the solutions

In this section, we consider the canonical basis $(\vec{e_1},\ldots,\vec{e_n})$ of Z^n and the j_i as the coordinates in this basis. The application $f(j_1,\ldots,j_n)=a_1j_1+\ldots+a_nj_n$ can be viewed as a linear transformation of the lattice Z^n. We first construct a matrix $N\in Z^n$ similar to the Hermite reduced form [S86] of that transformation:

$$(a_1,\ldots,a_n)=(10\ldots0)N$$

and N is an $n\times n$ unimodular matrix, i.e., $det(N)$ equals 1 or -1. Now we consider the new basis $(\vec{f_i})$ defined by $f_i=N^{-1}e_i$ and we call t_i the coordinates in this basis. The t_i are related to j_i by:

$$\begin{pmatrix}t_1\\t_2\\\vdots\\t_n\end{pmatrix}=N\begin{pmatrix}j_1\\j_2\\\vdots\\j_n\end{pmatrix}$$

In this new basis, f has a very simple form, since $f(j_1,\ldots,j_n)=f(N^{-1}(t_1,\ldots,t_n))=(10\ldots0)NN^{-1}(t_1,\ldots,t_n)$ t_1, and the number of solutions of equation $a_1j_1+\ldots+a_nj_n=C$ is exactly equal to the number of solutions of equation $t_1=C$, i.e., the number of integer points in the intersection of $t_1=C$ with the *iteration space*.

The *iteration space* considered is the infinite quadrant $j_1\geq 0,\ldots,j_n\geq 0$. Its intersection with the hyperplane $t_1=C$ is a simplex polyhedron with n vertices $A_1,\ldots,A_n$. A_k is the vertex which $(j_k)_{1\leq k\leq n}$ coordinates are equal to $(0,\ldots,0,\frac{1}{a_k}(C),0,\ldots,0)$, from which we can derive the t_i coordinates $\left(C,\frac{m_{2k}C}{a_k},\ldots,\frac{m_{nk}C}{a_k}\right)$.

Now, we work in the $(n-1)$ dimensional hyperplane $t_1=C$, for which $(\vec{f_2},\ldots,\vec{f_n})$ is a basis. In this basis, the volume of the polyhedron is given by the following expression:

$$\begin{aligned}V&=\frac{1}{(n-1)!}det\left(A_1\vec{A}_2,A_1\vec{A}_3,\ldots,A_1\vec{A}_n\right)\\&=\frac{1}{(n-1)!}C^{n-1}\begin{vmatrix}\frac{m_{22}}{a_2}-\frac{m_{21}}{a_1}&\cdots&\frac{m_{2n}}{a_n}-\frac{m_{21}}{a_1}\\\frac{m_{32}}{a_2}-\frac{m_{31}}{a_1}&&\vdots\\\vdots&&\\\frac{m_{n2}}{a_2}-\frac{m_{n1}}{a_1}&\cdots&\frac{m_{nn}}{a_n}-\frac{m_{n1}}{a_1}\end{vmatrix}\\&=\frac{C^{n-1}}{(n-1)!}\frac{d_{1\ldots n}}{a_1\ldots a_n}\end{aligned}$$

(This formula gives the (n-1)-dimensional volume of the "hypertetrahedron" defined as the space in the first coordinates quadrant, under the (n-2)-dimensional hyperplane defined by the $n-1$ vertices A_2, A_3, ..., A_n, where A_1 is taken as the origin.) The approximation lays in the fact that the number of integer points within a polyhedron is not exactly equal to its volume.

Erhart [E64] asserts that it is possible to give an exact formula for the number of integer points inside a polyhedron as soon as its vertices are integer points themselves. This requires that the $(j_i)_{1\leq i\leq n}$ or the $(t_i)_{1\leq i\leq n}$ coordinates of the vertices are integers, i.e., that $\forall i \in \{1,\ldots,n\}, a_i|C$. However, this is not likely to happen very often. Nevertheless, if it is possible to derive a fast method for computing the convex hull of any rational polyhedron, such a formula may then prove to be very useful.

3 Extracting the solutions

In this section, we explain how the solutions can be explicitly computed. Throughout this section, the constraints are those of Problem 2.1.

For each case (dimension 2, dimension 3, and dimension n), we will first describe the shape of the solutions for $(j_i)_{1\leq i\leq n}$ unbounded, and then derive the restrictions that must be imposed on the solutions to fit in the iteration space. Finally some means are given to enumerate the whole solution set, depending on the parameters used for describing the solutions.

3.1 Shape of the solutions for unbounded variables and dimensions 2 and 3

Because the Hermite reduced matrix can be used to generate the solutions easily, we refer to section 2.3.3 for the general case of generating the solutions. In this section we explicitly compute the Hermite reduced matrix N for the dimensions 2 and 3.

Case $n = 2$ In section 2.2.1, we gave a description of the solutions for $d_{12} = 1$. Let us now give the solutions for the general case ($d_{12} \neq 1$), since we will use it extensively in the rest of this section. Provided that C is a multiple of d_{12}, the solutions of equation (12) are:

$$\begin{aligned} j_1 &= Cu_1/d_{12} - (a_2/d_{12})\lambda \\ j_2 &= Cu_2/d_{12} + (a_1/d_{12})\lambda \end{aligned}$$

where λ is an integer parameter and u_1 and u_2 are integers verifying: $a_1u_1 + a_2u_2 = d_{12}$

Case $n = 3$ For $t_1 = C$, the solutions can be described by:

$$\begin{pmatrix} j_1 \\ j_2 \\ j_3 \end{pmatrix} = M \begin{pmatrix} t_1 \\ t_2 \\ t_3 \end{pmatrix} \tag{20}$$

with M unimodular and thus invertible. Its inverse is:

$$M^{-1} = \begin{pmatrix} a_1 & a_2 & a_3 \\ a_2u_2 + a_3u_3 & -a_2u_1 & -a_3u_1 \\ 0 & -v_3 & v_2 \end{pmatrix}$$

3.2 Extracting the solutions for bounded variables

We essentially have two methods for extracting the solutions: enumerate either the $(j_i)_{1\leq i\leq n}$ or the $(t_i)_{1\leq i\leq n}$ n-tuplets. On one hand, it is relatively easy to compute the bounds of the j_i parameters, but enumerating the $(j_i)_{1\leq i\leq n}$ n-tuplets is not very efficient since many such n-tuplets do not correspond to a solution; therefore many n-tuplets are unsuccessfully considered before being dropped. On the other hand, the space of solution of the t_i is much more dense since each of the t_i vary with stride 1 in this space. However, computing the bounds of these parameters is relatively costly.

When the number of the t_i variables is less or equal than the number of equations, a straightforward application of Gaussian Elimination can be used for isolating one of the variables, of which the obtained bounds can be used to recursively compute the bounds for the remaining variables. However, when the number of variables is larger than the number of equations, then this technique will not work. Therefore we resort to the Fourier pair-wise elimination method [D74]. This method succeeds by successive elimination

of each of the variables. In the following paragraphs we explain this technique for $n = 3$. It can be applied straightforward to the general case.

Recall the expression for the three dimensional case of $(j_i)_{1\leq i\leq 3}$ as functions of t_2, t_3 ($t_1 = C$). Since $j_i \in [M_i, N_i]$, it follows that:

$$M_1 \leq \quad u_1C + t_2 \quad \leq N_1 \tag{21}$$
$$M_2 \leq \quad u_2C - a_1v_2t_2 - a_3t_3 \quad \leq N_2 \tag{22}$$
$$M_3 \leq \quad u_3C - a_1v_3t_2 + a_2t_3 \quad \leq N_3 \tag{23}$$

The principle of the method is to isolate one parameter in each inequality, e.g. t_3. (For simplicity's sake we will assume that the a_i are non-negative in this example.)

$$M_1 \leq \quad u_1C + t_2 \quad \leq N_1 \tag{24}$$
$$\frac{1}{a_3}(u_2C - a_1v_2t_2 - N_2) \leq \quad t_3 \quad \leq \frac{1}{a_3}(u_2C - a_1v_2t_2 - M_2) \tag{25}$$
$$\frac{1}{a_2}(-u_3C + a_1a_3v_3t_2 - M_3) \leq \quad t_3 \quad \leq \frac{1}{a_2}(-u_3C + a_1a_3v_3t_2 - N_3) \tag{26}$$

Inequalities (25) and (26) give the interval of variation for t_3 (from the maximum of lower bounds up to the minimum of upper bounds).

Now, we eliminate t_3 by using the fact that any lower bound on t_3 must be less than any upper bound. Since this fact is trivially verified for the lower and upper bound of the same inequality, the following inequalities can be written:

$$M_1 \leq \quad u_1C + t_2 \quad \leq N_1 \tag{27}$$
$$\frac{1}{a_3}(u_2C - a_1v_2t_2 - N_2) \quad \leq \quad \frac{1}{a_2}(-u_3C + a_1a_3v_3t_2 - N_3) \tag{28}$$
$$\frac{1}{a_2}(-u_3C + a_1a_3v_3t_2 - M_3) \quad \leq \quad \frac{1}{a_3}(u_2C - a_1v_2t_2 - M_2) \tag{29}$$

from which we derive, by now isolating t_2 in turn:

$$M_1 - u_1C \quad \leq t_2 \leq \quad N_1 - u_1C \tag{30}$$
$$\frac{1}{a_1}((a_2u_2 + a_3u_3)C + a_3N_3 - a_2N_2) \quad \leq t_2 \leq \quad \frac{1}{a_1}((a_2u_2 + a_3u_3)C + a_3M_3 - a_2M_2) \tag{31}$$

This gives the interval of t_2. Note that two inequalities (28) and (29) are used to generate one two-sided inequality (31). In general n inequalities of the form of (28) and (29) can be used to generate $\lfloor n/2 \rfloor$ two-sided inequalities.

These formulas can now be used to generate the solutions of $a_1j_1 + a_2j_2 + a_3j_3 = C$ by the following DO loop structure:

$t_1 = C$
DO 1 $t_2 = max(M_1 - u_1C, \frac{1}{a_1}((a_2u_2 + a_3u_3)C + a_3N_3 - a_2N_2)),$
$\quad min(N_1 - u_1t_1, \frac{1}{a_1}((a_2u_2 + a_3u_3)C + a_3M_3 - a_2M_2)$
$\quad$ DO 1 $t_3 = max(\frac{1}{a_3}(u_2C - a_1v_2t_2 - N_2), \frac{1}{a_2}(-u_3C + a_1a_3v_3t_2 - M_3))$,
$\quad\quad min(\frac{1}{a_3}(u_2C - a_1v_2t_2 - M_2), \frac{1}{a_2}(-u_3C + a_1a_3v_3t_2 - N_3))$
$\quad\quad \ldots$
1 CONTINUE

and references to j_i within the loop body are replaced by references to t_i.

Note that for this example also Gaussian Elimination could have been used to isolate the variables, because the number of variables equals the number of equations, i.e. three. It can be easily verified that the resulting bounds are the same.

The Fourier-Motzkin method is known to be very expensive since it may generate an exponential number of equations. In our case, two facts make it tractable. The first one is that the dimension n of the problem is not large, and the second one is that, because of the particular form of the expressions, the number of inequalities generated seems to remain bounded. Moreover, the general expression of the bounds of $(t_i)_{1\leq i\leq n}$ can still be symbolically computed once for all, as it was shown here for the case $n = 3$. Then, the values of the coefficients can be substituted at compile-time or at run-time.

Equation #	N	C	a_1	M_1	N_1	a_2	M_2	N_2	a_3	M_3	N_3	a_4	M_4	N_4
1.2	2	-2	3	1	100	-5	1	50						
2.2	2	-6	-3	1	100	2	1	100						
3.2	2	129	-1	1	129	2	1	129						
4.2	2	0	1	1	129	8256	0	0						
5.2	2	0	1	1	129	4128	0	1						
6.2	2	0	1	1	129	2064	0	3						
7.2	2	128	128	0	0	129	1	64						
8.2	2	128	64	0	1	129	1	64						
9.2	2	26	23	-10	23	34	-9	12						
10.2	2	216	23	-110	23	134	-19	12						
1.3	3	0	-129	0	63	-1	2	64	129	0	63			
2.3	3	-15	4	1	100	2	1	50	-5	1	50			
3.3	3	374216	4	-110	23	256	-189	182	1024	-8713	2827			
4.3	3	374216	2	1999	2348	6	13318	18233	12	-8713	282737			
5.3	3	1000	1	-100	100	2	-100	100	3	-100	100			
1.4	4	110	-2	1	50	1	1	100	1	1	100	1	1	50
2.4	4	2	4	1	100	-2	1	100	-1	1	100	-2	1	100
3.4	4	12	-3	20	134	-1	1	200	1	20	134	2	1	200
4.4	4	1	1	1	100	2	1	50	3	1	100	5	1	50
5.4	4	-126	-129	0	63	-1	0	62	-1	0	62	129	0	63
6.4	4	100	2	-10	10	4	-10	10	8	-10	10	16	-10	10
7.4	4	12387	5	-18	130	78	-23	110	93	-10	180	134	-11	81

Table 1: Diophantine equations used as tests

4 Applications

To give an insight at the range of applications that could benefit from the results presented in the previous sections, we propose a set of possible applications, all related to the initial problem, that is, the explicit management of d–dimensional arrays within Fortran DO Loops.

Dependence test There already exists a number of dependence tests (GCD test, Banerjee [B88] test) which all require very few computations. However, none of these tests is exact for any value of $(a_i)_{1\leq i\leq n}$, $(M_i, N_i)_{1\leq i\leq n}$ and C. (In [PKP90], it is shown that Banerjee's test is exact only under certain conditions). Therefore, it is possible to write a heuristic which would first apply one of these tests and, if it fails or cannot be applied, would compute the exact number of solutions, as proposed in this paper. Since the number of operations required can be known *a priori*, it is possible to create a cost-function that would determine whether it is worth doing this computation, therefore, this is a "risk-free" technique. It must also be recalled that the determination of the number of solutions presented in this paper can be used for any value of the parameters. The efficiency of this test compared to the more standard tests is shown in the next section.

Parallelizing loops with dependencies Whenever dependence tests state that a loop has dependencies then it is generally executed serially. In the case of constant dependencies, it has been shown that it is possible to extract independent sets of iterations which can then be parallelized. However, when the dependencies are not constant, the parallelization of the loop is much harder.

In this paper, we showed how to extract the solutions of any dependence equation. So, it is possible to use this tool for getting the independent set of iterations that could then be executed concurrently. What is more, computing first the number of dependencies could give a hint at the number of independent sets that could be selected and therefore help decide if the whole operation is worth being performed.

Studying the locality of arrays Whenever an array has non-trivial subscripts, all its elements will not be accessed the same number of times. Therefore, it is interesting to get the "locality map" of the array, that is the number of references per each element. This is equivalent to solving equation (2) for a range of values of C. Ideally, we would like the number of solutions as a function of C. This is typically the role of the approximate functions presented in section 2.3, since these functions can be studied analytically and provide us with trends on the number of solutions.

Then, such information could be used in *data partitioning* for instance, to move those elements of matrices which are most frequently used in the upper parts of a memory hierarchy, see also [EJWB90].

5 Experimental Results

The following tables give a description of the experiments conducted and the results obtained. Basically, 25 Diophantine equations have been gathered from different sources (Perfect Club codes [Perf89], Banerjee's examples [B88], generic equations). For each equation, GCD-Banerjee's test, classic enumeration (i.e, test of each possible value of the j_i), Fast Determination technique, approximation of Fast Determination and Erhart approximation, have each been applied and timed on a Sparc 1+ workstation (optimization turned on).

	Banerjee - GCD		Enumeration		Fast Determination		Approx. of Fast Determination		Erbart Approx.	
Equation #	Sol.	Time	Nb sol.	Time	Nb sol.	Time	Nb sol.	Time	Nb sol.	Time
1.2	?	2e-4	17	5.3e-3	17	8e-4				
2.2	?	2e-4	33	1.3e-2	33	8e-4				
3.2	Yes	2e-4	65	2.0e-2	65	8e-4				
4.2	No	2e-4	0	7.0e-4	0	7e-4				
5.2	No	2e-4	0	7.0e-4	0	7e-4				
6.2	No	2e-4	0	7.0e-4	0	6e-4				
7.2	No	2e-4	0	7.0e-4	0	7e-4				
8.2	No	2e-4	0	7.0e-4	0	7e-4				
9.2	?	3e-4	1	1.9e-3	1	9e-4				
10.2	?	3e-4	0	7.1e-3	0	1.4e-3				
Avg.2	***	2.2e-4	6	5.1e-3	6	7.7e-4	***	***	***	***
1.3	Yes	3e-4	0	0.5	0	1.4e-3	31	2.3e-3	32	1.9e-3
2.3	?	3e-4	455	0.2	455	1.5e-3	457	2.3e-3	461	1.8e-3
3.3	?	3e-4	186	> 100	186	1.5e-3	0	2.3e-3	0	1.8e-3
4.3	?	3e-4	287586	> 100	287586	1.6e-3	286932	2.1e-3	282131	1.7e-3
5.3	No	3e-4	0	95.5	0	3.6e-3	0	2.5e-3	0	2.0e-3
Avg.3	***	3e-4	128	59	128	1.9e-3	92	2.3e-3	99	1.8e-3
1.4	Yes	3e-4	152760	45.4	152760	2.4e-2	151960	7e-3	152760	4.0e-3
2.4	Yes	3e-4	238625	84.6	238625	1.9e-2	238314	6e-3	238789	4.0e-3
3.4	Yes	3e-4	1121998	> 100	1121998	2.1e-2	1114207	8e-3	1121998	5.0e-3
4.4	No	3e-4	0	6e-4	0	1.6e-2	0	5e-3	0	5.0e-3
5.4	Yes	3e-4	0	> 100	0	2.7e-2	0	5e-3	1955	4.0e-3
6.4	?	3e-4	1067	2.1	1067	2.2e-2	0	7e-3	0	4.0e-3
7.4	?	3e-4	16766	> 100	16766	1.3e-1	16389	1e-3	16767	1.0e-3
Avg.4	***	3e-4	216350	62.5	216350	3.7e-2	214926	5.6e-3	216500	3.9e-3

Table 2: Results and timings (CPU time in seconds) of the different methods

As can be seen from these tables for classic enumeration the times can be substantially large (in some cases more than 100 seconds) depending on how many solutions exists. The surprising result is that fast determination is in most cases just slightly slower than GCD-Banerjee's test, a factor of 2 to 5.

Also the improvement in efficiency of the two approximation techniques over fast determination is at most a factor of 10, except for one case: equation 7.4. This would imply that except when efficiency is crucial for the data dependence analysis fast determination should be used in favor of the approximation techniques.

As can be expected the accuracy of the approximation techniques increases when the number of solutions becomes large. It should be noted that the approximation techniques do not lend themselves to decide whether there exists a solution if the number of solutions given by these approximation techniques is small.

6 Conclusion

In this paper we presented several sets of mathematical tools for characterizing the solutions of linear Diophantine equations. These equations occur frequently when studying memory references appearing in regular DO Loop structures.

The first set of tools aims at simplifying the constraints in order to reduce the complexity of the methods and to allow existing number theory results to be applied. The second set proposes three different methods for getting rapidly the exact number of solutions of Diophantine equations. Those methods show a tradeoff between the complexity of the computations required and their range of applicability. The third set of tools is composed of two approximations which provide a fast way for getting an approximation of the number of solutions, and also allow an analytical study of the closed expression for the number of solutions. Those two approximations have different range of applicability and error precision. The last tool proposed is a method for extracting the solutions themselves.

The main assets of these tools are their efficiency and applicability. Indeed, most of them can be used *for all cases*, independently of the value of the parameters and their time complexity is small enough to allow a compile-time and above all, ***run-time*** use, a fact which extends the scope of possible applications.

Though these tools can be taken as results in themselves, they should rather be considered as means for aiding data dependence analysis for restructuring compilers. Our goal was to stress the assets of our methods for dealing with Diophantine equations, rather than delivering complete applications, or solutions for given problems. Nevertheless, it is our belief that some applications based on these results, such as those presented in section 4, are worth further developments.

Acknowledgments

We would like to thank M. Crouzeix, W. Jalby and P. Kars for their helpful comments and suggestions.

References

[B88] U. Banerjee: *Dependence analysis for supercomputing*, Kluwer Academic Publisher, Norwell, Mass., 1988.

[C80] Comtet: *Combinatoric analysis*, PUF, 1980.

[D74] R. J. Duffin: *On Fourier's Analysis of Linear Inequality Systems*, North-Holland, 1974.

[E64] E. Erhart: *Sur un problème de géometrie Diophantienne linéaire (On a problem of linear diophantine geometry)*, June 1964, PhD Thesis at the university of Strassbourg, France.

[EJWB90] C. Eisenbeis, W. Jalby, D. Windheiser, F. Bodin: *A strategy for array management in local memory*, 1991, Advances in Languages and Compilers for Parallel Processing, MIT Press.

[GKP89] R. L. Graham, D. E. Knuth, O. Patashnik: *Concrete mathematics*, 1989, Addison-Wesley.

[Perf89] L. Pointer: *Perfect Club Report*, July 1989, CSRD Report No. 896.

[PS64] G. Polya, G. Szego: *Lessons and exercises of analysis*, 1964, Springer-Verlag.

[N72] M. Newman: *Integral Matrices*, Academic Press, London, 1972.

[PKP90] K. Psarris, X. Kong, D. Klappholz: *The I test: A New Test for Subscript Data Dependence*, Proceedings of 1990 International Conference on Parallel Processing, August 1990.

[S86] A. Schrijver: *Theory of Linear and Integer Programming*, Wiley-Interscience, 1986.

[ShLY89] She, Zhiyu, Zhiyuan Li and P-C. Yew: *An empirical study on array Subscripts and Data Dependencies*, August 1989, CSRD Report No. 840.

A Appendix

A.1 Modifying the complexity of the sums

The transformations below aim at modifying the complexity of the sums presented in section 2.2.1.

Case $n = 3$:

Let us now demonstrate these transformations for $n = 3$. It can be noticed that the following sum:

$$\Delta_\infty(a_1,a_2,a_3;C) = \sum_{\lambda=0}^{\lfloor\frac{C-a_3j_3^0}{a_3d_{12}}\rfloor} \lfloor\frac{(C-a_3j_3^0-\lambda d_{12}a_3)u_1}{a_2}\rfloor - \lceil\frac{-(C-a_3j_3^0-\lambda d_{12}a_3)u_2}{a_1}\rceil + 1$$

can be split into the following three sums:

$$\Delta_\infty(a_1,a_2,a_3;C) = \sum_{\lambda=0}^{\lfloor\frac{C-a_3j_3^0}{a_3d_{12}}\rfloor} \lfloor\frac{(C-a_3j_3^0-\lambda d_{12}a_3)u_1}{a_2}\rfloor - \sum_{\lambda=0}^{\lfloor\frac{C-a_3j_3^0}{a_3d_{12}}\rfloor} \lceil\frac{-(C-a_3j_3^0-\lambda d_{12}a_3)u_2}{a_1}\rceil + (\lfloor\frac{C-a_3j_3^0}{a_3d_{12}}\rfloor + 1)$$

Let us now concentrate on the first sum; all transformations performed on this sum can be applied similarly to the second one:

$$\sum_{\lambda=0}^{\lfloor\frac{C-a_3j_3^0}{a_3d_{12}}\rfloor} \lfloor\frac{(C-a_3j_3^0-\lambda d_{12}a_3)u_1}{a_2}\rfloor = \sum_{\lambda=0}^{\lfloor\frac{A_1}{B_1}\rfloor} \lfloor\frac{A_1-B_1\lambda}{C_1}\rfloor$$

with $A_1 = (C - a_3j_3^0)u_1$, $B_1 = d_{12}a_3u_1$ and $C_1 = a_2$.

Parameter λ can be decomposed as follows, using the Euclidean division by a_2:

$$\lambda = Q_2a_2 + R_2$$

Then the previous sum can be decomposed itself into two sums, with Q_2 and R_2 as new parameters. Since R_2 is the remainder of a division by a_2, then $0 \leq R_2 \leq min(C_1 - 1, \lfloor\frac{A_1}{B_1}\rfloor)$. Since $0 \leq \lambda \leq \lfloor\frac{A_1}{B_1}\rfloor$, then $0 \leq Q_2 \leq \lfloor\frac{\lfloor\frac{A_1}{B_1}\rfloor - R_2}{C_1}\rfloor$.

$$\sum_{\lambda=0}^{\lfloor \frac{A_1}{B_1} \rfloor} \lfloor \frac{A_1 - B_1\lambda}{C_1} \rfloor$$

$$= \sum_{R_2=0}^{min(C_1-1, \lfloor \frac{A_1}{B_1} \rfloor)} \sum_{Q2=0}^{\lfloor \frac{\lfloor \frac{A_1}{B_1} \rfloor - R_2}{C_1} \rfloor} \lfloor \frac{A_1 - B_1C_1Q_2 - B_1R_2}{C_1} \rfloor$$

$$= \sum_{R_2=0}^{min(C_1-1, \lfloor \frac{A_1}{B_1} \rfloor)} \sum_{Q2=0}^{\lfloor \frac{\lfloor \frac{A_1}{B_1} \rfloor - R_2}{C_1} \rfloor} \lfloor \frac{A_1 - B_1R_2}{C_1} \rfloor - B_1Q_2$$

$$= \sum_{R_2=0}^{min(C_1-1, \lfloor \frac{A_1}{B_1} \rfloor)} \lfloor \frac{\lfloor \frac{A_1}{B_1} \rfloor - R_2}{C_1} \rfloor \times \lfloor \frac{A_1 - B_1R_2}{C_1} \rfloor - \frac{\lfloor \frac{\lfloor \frac{A_1}{B_1} \rfloor - R_2}{C_1} \rfloor \times (\lfloor \frac{\lfloor \frac{A_1}{B_1} \rfloor - R_2}{C_1} \rfloor + 1)}{2}$$

Remarks

- The terms of the sum are of the same complexity as in the original sum, and the order of this new sum depends on a coefficient only.

Case $n = 4$:

The transformations are identical for $n = 4$.

$$\Delta_\infty(a_1, a_2, a_3, a_4; C) =$$
$$\sum_{\lambda=0}^{\lfloor \frac{C - d_{12}\mu^0}{d_{12}d_{34}} \rfloor} (\lfloor \frac{(\mu^0 d_{12} + \lambda d_{12}d_{34})u_1}{a_2} \rfloor - \lceil \frac{-(\mu^0 d_{12} + \lambda d_{12}d_{34})u_2}{a_1} \rceil + 1) \times (\lfloor \frac{(C - \mu^0 d_{12} - \lambda d_{12}d_{34})u_3}{a_4} \rfloor - \lceil \frac{-(C - \mu^0 d_{12} - \lambda d_{12}d_{34})u_4}{a_3} \rceil + 1)$$

This sum can be subdivided into four sums. For each sum, the transformations are similar to the case $n = 3$.

A.2 Erhart polynomials

Let us assume that $C' = C + \frac{\sum_{i=1}^n a_i}{2}$

$n = 2$:

$$f(C) = \frac{C'}{\prod_{i=1}^n a_i}$$

$n = 3$:

$$f(C) = \frac{C'^2 - \frac{\sum_{i=1}^n a_i^2}{12}}{2\prod_{i=1}^n a_i}$$

$n = 4$:

$$f(C) = \frac{C'^3 - \frac{\sum_{i=1}^n a_i^2}{4} C'}{6\prod_{i=1}^n a_i}$$

$n = 5$:

$$f(C) = \frac{C'^4 - \frac{\sum_{i=1}^n a_i^2}{2} C'^2}{24\prod_{i=1}^n a_i} + \frac{\frac{1}{24}\left[\frac{(\sum_{i=1}^n a_i^2)^2}{2} + \frac{\sum_{i=1}^n a_i^4}{5}\right]}{24\prod_{i=1}^n a_i}$$

$n = 6$:

$$f(C) = \frac{C'^5 - 5\frac{\sum_{i=1}^n a_i^2}{6} C'^3}{120\prod_{i=1}^n a_i} + \frac{\frac{5}{24}\left[\frac{(\sum_{i=1}^n a_i^2)^2}{2} + \frac{\sum_{i=1}^n a_i^4}{5}\right]}{120\prod_{i=1}^n a_i} C'$$

A.3 Approximations of section 2.3.1

Case $n = 3$:

Let us now demonstrate these transformations for $n = 3$. It can be noticed that the following sum:

$$\Delta_\infty(a_1,a_2,a_3;C) = \sum_{\lambda=0}^{\lfloor\frac{C-a_3j_3^0}{a_3d_{12}}\rfloor} \lfloor\frac{(C-a_3j_3^0-\lambda d_{12}a_3)u_1}{a_2}\rfloor - \lceil\frac{-(C-a_3j_3^0-\lambda d_{12}a_3)u_2}{a_1}\rceil + 1$$

can be split into the following three sums:

$$\Delta_\infty(a_1,a_2,a_3;C) = \sum_{\lambda=0}^{\lfloor\frac{C-a_3j_3^0}{a_3d_{12}}\rfloor} \lfloor\frac{(C-a_3j_3^0-\lambda d_{12}a_3)u_1}{a_2}\rfloor - \sum_{\lambda=0}^{\lfloor\frac{C-a_3j_3^0}{a_3d_{12}}\rfloor} \lceil\frac{-(C-a_3j_3^0-\lambda d_{12}a_3)u_2}{a_1}\rceil + (\lfloor\frac{C-a_3j_3^0}{a_3d_{12}}\rfloor + 1)$$

Let us now concentrate on the first sum; all transformations performed on this sum can be applied similarly to the second one:

$$\sum_{\lambda=0}^{\lfloor\frac{C-a_3j_3^0}{a_3d_{12}}\rfloor} \lfloor\frac{(C-a_3j_3^0-\lambda d_{12}a_3)u_1}{a_2}\rfloor = \sum_{\lambda=0}^{\lfloor\frac{A_1}{B_1}\rfloor} \lfloor\frac{A_1-B_1\lambda}{C_1}\rfloor$$

with $A_1 = (C - a_3j_3^0)u_1$, $B_1 = d_{12}a_3u_1$ and $C_1 = a_2$.

Now, by replacing $\lfloor\alpha\rfloor$ by $\alpha - \frac{1}{2}$, we obtain:

$$\begin{aligned}
&\sum_{\lambda=0}^{\lfloor\frac{A_1}{B_1}\rfloor} \lfloor\frac{A_1-B_1\lambda}{C_1}\rfloor \\
&= \sum_{\lambda=0}^{\lfloor\frac{A_1}{B_1}\rfloor} \frac{A_1-B_1\lambda}{C_1} - \frac{1}{2} \\
&= (\frac{A_1}{C_1} - \frac{1}{2}) \times \lfloor\frac{A_1}{B_1}\rfloor - \frac{B_1}{C_1}\frac{(\lfloor\frac{A_1}{B_1}\rfloor)(\lfloor\frac{A_1}{B_1}\rfloor+1)}{2}
\end{aligned}$$

Case $n = 4$:

The transformations are identical for $n = 4$.

$$\begin{aligned}
&\Delta_\infty(a_1,a_2,a_3,a_4;C) \\
&= \sum_{\lambda=0}^{\lfloor\frac{C-d_{12}\mu^0}{d_{12}d_{34}}\rfloor} (\lfloor\frac{(\mu^0d_{12}+\lambda d_{12}d_{34})u_1}{a_2}\rfloor - \lceil\frac{-(\mu^0d_{12}+\lambda d_{12}d_{34})u_2}{a_1}\rceil + 1) \times (\lfloor\frac{(C-\mu^0d_{12}-\lambda d_{12}d_{34})u_3}{a_4}\rfloor - \lceil\frac{-(C-\mu^0d_{12}-\lambda d_{12}d_{34})u_4}{a_3}\rceil + 1) \\
&\simeq (\frac{(\mu^0d_{12}+\lambda d_{12}d_{34})u_1}{a_2} - \frac{-(\mu^0d_{12}+\lambda d_{12}d_{34})u_2}{a_1} + 1) \times (\frac{(C-\mu^0d_{12}-\lambda d_{12}d_{34})u_3}{a_4} - \frac{-(C-\mu^0d_{12}-\lambda d_{12}d_{34})u_4}{a_3} + 1)
\end{aligned}$$

LISTE DES PUBLICATIONS INTERNES IRISA 1992

PI 624 SIGNAL AS A MODEL FOR REAL-TIME AND HYBRID SYSTEMS
Albert BENVENISTE, Michel LE BORGNE, Paul LE GUERNIC
Janvier 1992, 22 pages.

PI 625 ON THE CENTRAL-LIMIT THEOREM FOR TRACKING ESTIMATORS WITH SMALL GAIN - INFINITE HORIZON CASE
Bernard DELYON, Anatoli JUDITSKY
Janvier 1992, 16 pages.

PI 626 A MONTE CARLO METHOD BASED ON ANTITHETIC VARIATES FOR NETWORK RELIABILITY COMPUTATIONS
Mohamed EL KHADIRI, Gerardo RUBINO
Janvier 1992, 28 pages.

PI 627 CONSTRAINED MULTISCALE MARKOV RANDOM FIELDS AND THE ANALYSIS OF VISUAL MOTION
Fabrice HEITZ, Patrick PEREZ, Patrick BOUTHEMY
Janvier 1992, 40 pages.

PI 628 ON ITERATIVE REFINEMENT FOR THE SPECTRAL DECOMPOSITION OF SYMMETRIC MATRICES
Alexander N. MALYSHEV
Janvier 1992, 26 pages.

PI 629 STRUCTURAL OPERATIONAL SPECIFICATIONS AND TRACE AUTOMATA
Eric BADOUEL, Philippe DARONDEAU
Janvier 1992, 36 pages.

PI 630 EREBUS, A DEBUGGER FOR ASYNCHRONOUS DISTRIBUTED COMPUTING SYSTEM
Michel HURFIN, Noël PLOUZEAU, Michel RAYNAL
Janvier 1992, 14 pages.

PI 631 PROTOCOLES SIMPLES POUR L'IMPLEMENTATION REPARTIE DES SEMAPHORES
Michel RAYNAL
Janvier 1992, 14 pages.

PI 632 L-STABLE PARALLEL ONE-BLOCK METHODS FOR ORDINARY DIFFERENTIAL EQUATIONS
Philippe CHARTIER, Bernard PHILIPPE
Janvier 1992, 28 pages.

PI 633 ON EFFICIENT CHARACTERIZING SOLUTIONS OF LINEAR DIOPHANTINE EQUATIONS AND ITS APPLICATION TO DATA DEPENDENCE ANALYSIS
Christine EISENBEIS, Olivier TEMAM, Harry WIJSHOFF
Janvier 1992, 22 pages.

PI 634 UN NOYAU DE SYSTEME REPARTI POUR LES APPLICATIONS GEREES PAR UN TEMPS VIRTUEL
Janvier 1992, 20 pages.

Imprimé en France
par
l'Institut National de Recherche en Informatique et en Automatique

ISSN 0249-6399